# 객체모델을 이용한, XML DTD의 ORDB 스키마로의 변환

이 상 태* · 주 경 수**

# Transformation from XML DTD to ORDB Schema using Object Model

Sang-Tae Lee* · Kyung-Soo Joo**

## Abstract

XML is a standard markup language for exchange and storage of formed or well-formed information in World Wide Web. Because the source data of XML message for exchange of information in World Wide Web is stored in legacy database, it is necessary for the easy connection between XML application and database system. In Oracle8i, 9i, Informix and SQL2000, DBMS vendors make upgrade to DBMS for using XML. This method of upgrade between XML application and database system is Platform-dependent and DBMS-dependent. Also It is necessary for the method of the platform- and DBMS-independent connection between XML application and database system.

The methods for the connection between XML DTD and RDB schema are studied for the easy connection between XML application and database system. But the study for the easy connection between XML DTD and ORDB schema is a little. For multimedia application, we use the extended DBMS from RDBMS. It is necessary for the study to the transformation from XML DTD to ORDB schema.

In this paper, for easier connection between XML application and database system, we propose the method of the transformation from XML DTD to ORDB schema using Object Model.

※ 본 연구는 정보통신부의 ITRC 사업에 의해 수행된 것임.
* 신성대학 컴퓨터응용계열 교수
** 순천향대학교 정보기술공학부 교수

## 1. 서 론

XML은 웹에서 구조화된 정보나 반-구조화된 정보를 교환하기 위한 표준 마크업 언어로 채택되어 가고 있다. 한편 웹에서 정보교환을 위한 XML 메시지의 소스 데이터는 Legacy 데이터베이스에 저장되어 있기 때문에, 이에 따라 XML 응용과 데이터베이스 시스템과의 원활한 연계가 요구되어진다. Oracle8i와 9i 및 Informix 그리고 SQL2000 등과 같이, DBMS 공급자들은 XML을 취급하기 위하여 자신의 DBMS들을 확장하고 있다. 이러한 방식의 XML 응용과 데이터베이스 시스템과의 연계방법은 플랫폼-종속적이며 아울러 DBMS-종속적이다. 따라서 보다 원활한 XML 응용과 데이터베이스 시스템과의 연계를 위해서는 플랫폼-독립적이며 DBMS-독립적인 연계방안이 요망된다.

XML 응용과 데이터베이스 시스템 사이의 원활한 연계를 위해서, XML DTD를 관계형 데이터베이스 스키마로 변환하는 방법에 대해서는 많은 연구가 진행되었으나, XML DTD를 객체-관계형 데이터베이스 스키마로 변환키 위한 연구는 미미한 실정이다. 멀티미디어 응용 등을 위하여 기존의 관계형 DBMS들이 객체-관계형 DBMS로 확장된 현실에서 XML DTD를 객체-관계형 데이터베이스 스키마로 변환키 위한 연구는 조속히 요구되고 있다.

XML의 태그들은 문서의 구조나 데이터의 의미를 나타내기 위해서 사용된다. 따라서 문서를 작성할 때 문서의 구조 혹은 데이터의 의미에 따라 사용자가 필요한 태그들을 자유롭게 정의할 수 있으며, DTD는 XML 문서 내에서 사용 가능한 요소를 관리하고, 사용할 수 있는 각 요소형의 내용과 특성을 지정하는 규칙들을 담고 있다[1]. 본 논문에서는 이 DTD에 정의한 요소와 특성들을 객체화하여 객체-관계형 데이터베이스 스키마로 변환하는 방법을 제안한다.

본 논문의 제2절에서는 객체-관계형 데이터베이스에 대하여 설명하고, 제3절에서는 객체 기반 변환 방법에 대하여 설명하며, 제4절에서는 XML DTD에서 객체로 변환하는 방법을 다루고, 제5절에서는 객체를 객체-관계형 데이터베이스 스키마로 변환하는 방법을 다루며, 마지막으로 결론을 기술한다.

## 2. 객체-관계형 데이터베이스

객체-관계형 데이터베이스는 객체들을 다루기 위하여 관계형 데이터베이스에서 확장되었으며, 아직도 완전한 표준화가 이루어져 있지 않다. 그러나 SQL3가 점차 표준으로 자리 잡아가고 있다.

객체-관계형 데이터베이스 시스템들은 복잡한 데이터도 처리할 수 있으며, DBMS 공급자들이 새로운 기술에 적용할 수 있기 때문에 유니버설 서버나 데이터베이스 관리자로 알려져 있다. 새로운 기술로 데이터를 처리하기 위해 확장된 기본적인 내용들은 아래와 같다.

(1) Large object data types
(2) Extended data types
(3) Multivalued data types
(4) Object identity
(5) Extensions that aid in using the above new fearures

먼저 객체-관계형 데이터베이스 시스템들은 복잡한 객체를 다루기 위하여 BLOBs, CLOBs, FLOBs 등의 새로운 타입을 추가 하였다. LOB는 큰 객체를 다루기 위한 것이며, 이러한 데이터 타입들은 SQL 데이터로 객체를 컴퓨터에서 다룰 수 있도록 하고 있다.

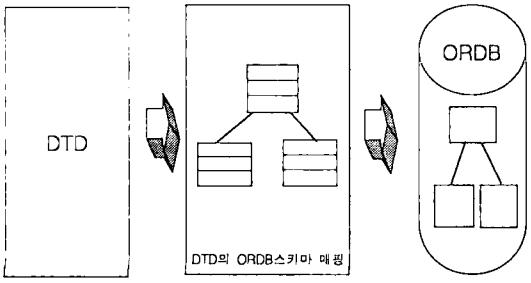
ANSI SQL3 표준은 세 개의 큰 객체 종류를 제공하고 있는데 그들은 아래와 같다.

(1) CLOB(Character Large OBject)
(2) NCLOB(National Character Large OBject)
(3) BLOB(Binary Large OBject)

위의 타입들이 사용자 정의 타입으로 될 수 없기 때문에 객체-관계형 데이터베이스 시스템들은 시스템에서 이러한 객체들이 다루어지도록 메소드를 첨가한다. 오라클8, Informix Dynamic Server, DB2 Universal Data Base(UDB)는 모두 이러한 타입을 지원하지만 서로 다른 문장 구조를 가지고 있다. 점차 관계형 데이터베이스에서 객체-관계형 데이터베이스로 전환되고 있는 환경에서 확장의 표준화가 요구된다.

## 3. 객체 기반 변환

XML DTD를 데이터베이스 스키마로 변환하고자 할 때 단순한 XML 문서는 직접 테이블로 변환이 가능하나 복잡한 XML 문서를 다룰 때는 객체 기반 변환 방법으로 처리되어야 한다. 따라서 테이블로 직접 변환하는 방법으로는 복잡한 XML 문서를 변환하기가 불가능하므로 이 단점을 개선시키기 위해 중간에 객체를 이용하여 (그림 1)과 같이 객체-관계형 데이터베이스 스키마로 변환한다[3].




(그림 1) 객체 기반 매핑

## 4. XML DTD의 객체 기반 매핑

### 4.1 요 소

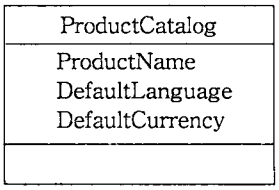
HTML은 태그가 직접 스타일 시트로 사용되는데 시작 태그는 성질을 열고, 끝 태그는 다시 닫는다. XML에서는 시작 태그와 끝 태그가 컨테이너로 사용되며 시작 태그의 내용과 끝 태그가 함께 하나의 요소를 이룬다. 요소는 하나의 루트 요소만 가져야 하며, 다른 태그는 모두 요소 안에 확실하게 중첩되어야 한다[2] [6]. 이것은 한 요소가 다른 요소들을 포함할 경우, 그 요소들은 한 요소 안에 들어가야 한다는 뜻이다. 요소 타입은 두 개의 타입으로 분류하는데 PCDATA만 가진 요소의 타입을 단순 요소 타입이라고 하며, PCDATA을 제외한 모든 요소의 타입을 복합 요소 타입이라고 한다. 즉 복합 요소 타입은 요소의 자식 요소, 혼합 요소 그리고 요소의 특성들이다.

단순 요소 타입을 객체로 변환할 때는 클래스의 속성으로 변환된다. 다음 예제는 단순 요소 타입의 예를 보여준다.

```
<!ELEMENT ProductCatalog (ProductName,
                DefaultLanguage, DefaultCurrency)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
```

(그림 2) 단순 요소 타입

(그림 2)를 클래스 속성으로 변환하면 (그림 3)과 같다.




(그림 3) 단순 요소 타입을 객체로 변환

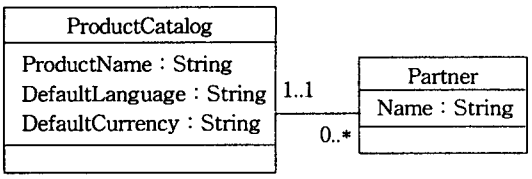
(그림 3)에서 보면 클래스 이름은 (그림 2)에 있는 요소가 포함하고 있는 부모 요소가 클래스 이름으로 지정되고, 데이터형은 작성자가 요소의 의미에 따라 정수형, 문자형 등으로 지정한다.

복합 요소 타입은 단순 요소 타입을 제외한 모든 요소를 말하는데 (그림 4)와 같다.

```
<!ELEMENT ProductCatalog (ProductName,
                DefaultLanguage, DefaultCurrency,
                Partner*)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Partner (Name)>
```

(그림 4) 복합 요소 타입

(그림 4)에서는 루트 요소가 ProductCatalog이고, 자식 요소는 ProductName, DefaultLanguage, DefaultCurrency, Partner이다. 따라서 객체로 변환하면 (그림 5)와 같다.




(그림 5) 복합 요소 타입의 객체로 변환

(그림 5)는 객체로 변환된 클래스이다. 클래스 이름은 ProductCatalog이다. 여기서 데이터형은 (그림 2)에서 요소의 의미가 문자형으로 사용되므로 객체로 변환될 때 String형으로 변환된다.

## 4.2 특 성

모든 요소는 특성을 가질 수 있다. 특성은 이름 또는 값의 형태를 가진다. 특성은 요소에 포함되며, 특성에 부여된 값을 통해서 그 요소에 어떠한 특징을 제공하게 된다[4] [5].

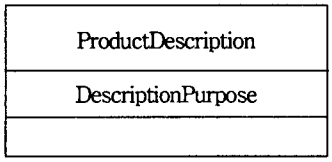
특성은 XML 파서가 애플리케이션에게 보내는 값들을 뜻하는데 요소의 내용과는 구별되는 값이므로 요소와 마찬가지고 특성도 단일 값을 가진 특성과 다중 값을 가진 특성으로 분류된다.

단일 값을 가진 특성은 문자열 특성(CDATA), 토큰 특성(ID, IDREF, NMTOKEN, ENTITY, NOTATION), 열거형 특성이며, 객체로 변환될 때 클래스의 속성으로 변환된다.

```
<!ELEMENT ProductDescription>
<!ATTLIST ProductDescription
            DescriptionPurpose CDATA #IMPLIED>
```

(그림 6) 단일 값을 가진 특성

(그림 6)에서 보면 클래스 이름이 ProductDescription이고 클래스의 속성은 DescriptionPurpose이다.




(그림 7) 단일 값을 가진 특성을 객체로 변환

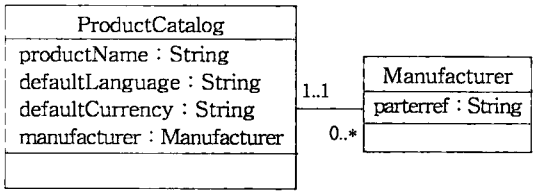
다중 값을 가진 특성은 IDREFS, NMTOKENS, ENTITIES으로 선언된 것이다. 이것들을 객체로 변환할 때 값이 하나 이상으로 들어가기 때문에 별도 의 객체로 만들 수 있다. (그림 8)은 특성을 포함한 DTD를 보여준다.

(그림 8)에서 보면 요소의 특성인 PartnerRef은 IDREFS로 선언되어 있다. IDREF는 단일 값을 가지고 있는 토큰 특성 ID와 의미는 같으나 ID는 요소에 대한 ID로 같은 문서에서 같은 ID 특성 값을 가진 두 개의 요소를 가질 수 없으며, IDREF는 ID를 가리키는 포인터이고, IDREFS는 하나 이상의 IDREF값으로 이루어진다. 따라서 객체로 만들 수 있다.

```
<!ELEMENT ProductCatalog (ProductName,
                DefaultLanguage, DefaultCurrency,
                                Manufacturer)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Manufacturer>
<!ATTLIST Manufacturer
                PartnerRef IDREFS #IMPLIED>
```

(그림 8) 다중 값을 가진 특성

(그림 9)에서는 두 개의 클래스로 이루어지며, 한 클래스는 부모 클래스이고 다른 하나는 자식 클래스로 구분된다. (그림 8)에서 부모 요소는 ProductCatalog이고 자식 요소는 Manufacturer이므로 객체로 변환하면 (그림 9)와 같다. 따라서 두 개의 클래스를 연결하려면 참조형으로 연결된다.




(그림 9) 복합 요소 타입의 객체로 변환

## 4.3 복합 내용 모델 변환

### 4.3.1 순 차

DTD에 있는 요소와 요소에 속하는 자식 요소들을 순차적으로 객체로 변환하면, 자식 클래스와 부모 클래스간의 연결을 참조형으로 변환할 수 있다. 테이블 스키마로 변환할 때 그 테이블이 속하는 컬럼들은 반드시 NOT NULL 제약조건이 따른다.

### 4.3.2 선 택

DTD 요소 안에 자식 요소들이 있는데 그 중에 하나만 선택할 경우에도 객체로 변환될 수 있고, 변환된 객체는 다시 객체-관계형 데이터베이스 스키마 객체 모델로 변환된다.

### 4.3.3 반 복

요소 안에 자식 요소가 중복된 경우 객체화로 변환할 때 요소 안에 중복된 같은 요소들이 클래스 속성으로 변환하는 경우 배열 형태로 변환된다. 이렇게 변환된 클래스 속성은 별도의 테이블로 변환된다. 하나 이상의 요소들이 존재한다는 의미로 클래스 속성의 데이터형을 String[]으로 변환한다. 즉 작성자가 알 수 없는 만큼 요소가 존재한다는 의미로 별도의 테이블로 변환된다.

### 4.3.4 서브그룹

< !ELEMENT A (B, (C | D)) >와 같은 경우를 말하며, 이 요소는 < !ELEMENT A (B, C) >와 <!ELEMENT A (B, D) > 요소로 분류된다. 따라서 두 개의 객체로 변환할 수 있다.

## 4.4 혼합 내용 모델 변환

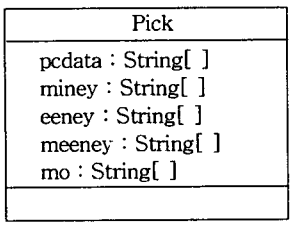
텍스트나 요소, 그 둘을 모두 포함할 수 있는 요소들을 혼합 내용 모델이라고 하며, XML 프로세서는 공백, 탭 등의 PCDATA와 요소 내용간의 구분이 어렵다. 왜냐하면 끝 태그와 다음의 시작 태그 사이에 공백이 있으면 불분명하게 된다. 혼합 내용 모델에서 선택이 가능한 그룹은 하나이고, #PCDATA로 시작하며, 그 뒤에는 혼합 내용의 연산자수를 나타내는 타입 순서로 되며, 각각은 한 번만 선언된다[7]. #PCDATA만 유일한 옵션이며, "*"는 반드시 괄호를 닫은 바로 뒤에 와야 한다. 다음은 혼합 내용 모델의 DTD 예를 보여준다.

```
<!ELEMENT Pick (#PCDATA | eeney |
                meeney | miney | mo)* )>
<!ELEMENT eeney (#PCDATA)>
<!ELEMENT meeney(#PCDATA)>
<!ELEMENT miney(#PCDATA)>
<!ELEMENT mo(#PCDATA)>
```

(그림 10) 혼합 내용 모델 DTD

(그림 10)은 혼합 내용 모델의 DTD이다. Pick 요소는 루트 요소이고 자식 요소들은 반복적으로 이루어진다. 따라서 루트 요소는 클래스 이름으로 변환하고 자식 요소는 클래스의 속성으로 변환된다. 그리고 반복성 때문에 데이터형은 배열로 선언된다. 이 DTD를 객체로 변환하면 (그림 11)과 같다.

DTD에서 "*" 연산자는 요소나 요소 그룹이 생략되거나 한 번 이상 나타날 수 있으므로 설계하는 작성자는 몇 개가 나타나는지 알 수 없기 때문에 String[] 형으로 선언한다.

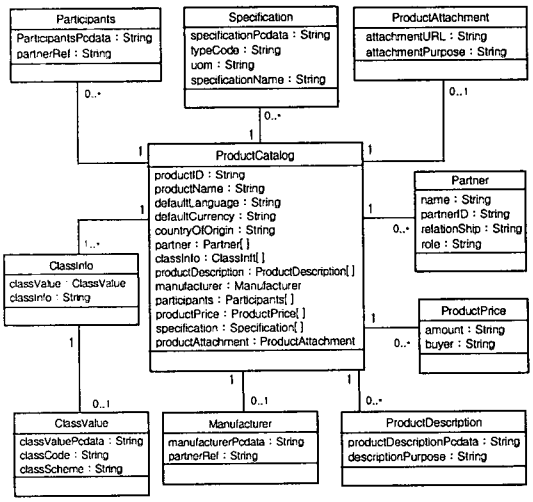



(그림 11) 혼합 내용 모델 DTD를 객체로 변환

위에 설명된 내용에 따라 (그림 12)에 보여준 XML DTD는 (그림 13)과 같이 객체모델로 변환될 수 있다.

```
<?xml version="1.0" encoding="EUC-KR"?>
<!ELEMENT ProductCatalog (ProductName,
                DefaultLanguage, DefaultCurrency,
        Partner*, ClassInfo+, ProductDescription*,
  CountryOfOrigin?, Manufacturer?, Participants*,
ProductPrice*, Specification*, ProductAttachment?)>
<!ATTLIST ProductCatalog ProductID ID
                                #REQUIRED>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Partner (Name)>
<!ATTLIST Partner
          PartnerID ID #IMPLIED
          RelationShip (Seller | Manufacturer |
                    Imtermediary) #IMPLIED
          Role CDATA #IMPLIED >
<!ELEMENT Name (#PCDATA)>
<!ELEMENT ClassInfo (ClassValue, ClassInfo?)>
<!ELEMENT ClassValue (#PCDATA)>
<!ATTLIST ClassValue
          ClassCode CDATA #IMPLIED
          ClassScheme CDATA #IMPLIED >
<!ELEMENT ProductDescription (#PCDATA)>
<!ATTLIST ProductDescription
          DescriptionPurpose CDATA
          #IMPLIED >
<!ELEMENT CountryOfOrigin (#PCDATA)>
<!ELEMENT Manufacturer (#PCDATA)>
<!ATTLIST Manufacturer
          PartnerRef IDREF #IMPLIED >
<!ELEMENT Participants (#PCDATA)>
<!ATTLIST Participants
          PartnerRef IDREF #IMPLIED >
<!ELEMENT ProductPrice (Amount, Buyer?)>
<!ELEMENT Amount (#PCDATA)>
<!ELEMENT Buyer (#PCDATA)>
<!ELEMENT Specification (#PCDATA)>
<!ATTLIST Specification
          TypeCode (Size | Weight | Ingredient |
          Color | Shape | Packing | Performance
          | Content | Others) #IMPLIED
          UOM CDATA #IMPLIED
      SpecificationName CDATA #REQUIRED>
<!ELEMENT ProductAttachment
(AttachmentURL, AttachmentPurpose?)>
<!ELEMENT AttachmentURL (#PCDATA)>
<!ELEMENT AttachmentPurpose (#PCDATA)>
```

(그림 12) XML DTD




(그림 13) 객체모델

## 5. 객체로부터 객체-관계형 데이터 베이스 스키마로 변환

### 5.1 변 환

XML DTD의 요소들을 객체클래스를 이용하여 객체-관계형 데이터베이스 스키마로 변환한다. 먼저 XML DTD의 요소들을 객체클래스로 변환하여 모델링하면 (그림 13)에서 나타나는 것과 같이 객체(Object), 관계성(Relationships), 멀티플리시티(Multiplicity), 방향성(Direction)이 만들어지며, 이들은 객체-관계형 데이터베이스 스키마 객체, 즉 객체 타입(Object Type), 객체 테이블(Object Table), 참조 대 포함(Referenced vs Embedded), 컬렉션(Collections)으로 변환된다[8] [9].

### 5.2 변환 과정

다음은 XML DTD로부터 변환된 객체클래스가 객체-관계형 데이터베이스 스키마로 변환하는 과정이다.

① 타입, 테이블, 보조적인 객체를 생성하기 위해서는객체-관계형 데이터베이스 시스템으로 확장되어 사용되며, 그러한 예로는 오라클8, Informix, SQL2000, DB2가 있다.

② 객체 클래스는 구조화된 객체 타입을 생성하며; 테이블과 연결하여 타입을 하나 또는 여러 개의 테이블에 객체의 원소가 되도록 한다.

③ 객체 타입은 사용자 정의 타입이거나, 속성을 가진 객체타입이다.

④ 클래스에서의 객체 속성은 객체 타입에서의 속성이다.

⑤ 타입이 테이블이나 객체 또는 특정 타입으로 변환할 때 클래스에서의 객체 속성 타입은 객체 타입에서의 속성 타입이다.

⑥ 객체 속성이 NULL 제약조건을 가지면 NOT NULL 제약조건도 가진다.

⑦ 객체 속성이 초기값을 가지면 컬럼에 DEFAULT 값을 더한다.

⑧ 독립적인 클래스나 함축적인 성질을 가진 클래들을 위해 기본키로 정수를 생성한다. {oid}를 위해 태그된 컬럼에 기본키 제약조건에 따라 {oid}를 더한다.

⑨ 부 클래스를 위해 기본키 제약조건과 외래키 제약조건에 따라 각각의 부모 클래스의 키를 더한다.

⑩ 클래스들의 결합을 위해 객체 타입을 생성하고 기본키 제약조건과 외래키 제약조건에 따른 역할 테이블로부터 기본키를 추가한다.

⑪ 교환 oid가 <n>인 경우 UNIQUE 제약조건에 따른 컬럼을 추가한다.

⑫ 각각의 확실한 제약조건을 위해 CHECK를 실시한다.

⑬ 결합하는데 있어서 각각의 0..1, 1..1 역할을 위한 참조 테이블에서 외래키 컬럼을 생성한다. 교대로 단일 객체들이나 컬럭션을 위한 그 자신 안에서 속성으로 객체를 선언하기 위하여 객체 타입에 참조를 사용하거나 또는 다중관계 객체를 위한 참조의 배열로 객체를 선언하기 위하여 객체 타입에 참조를 사용한다.

⑭ 집합 테이블에 외래키를 가진 다중 집합을 위해 기본키를 생성하고, 기본키를 위한 컬럼을 추가하며, 테이블 안에서 그 집합을 저장하기 위한 객체 타입을 사용한다. 또한 단일 객체를 위한 객체 속성 또는 컬렉션을 통하여 배열이나 네스티드 테이블을 사용한다.

⑮ 너무 많이 이동하게 되는 이진 결합 클래스들을 최적화 한다.

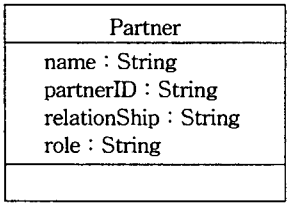
⑯ 외래키를 사용하여 결합이 안된 클래스와 함께 다대다 결합의 관계 테이블을 생성한다.

⑰ 다대다 결합에서 역할 테이블의 키로부터 기본키, 외래키의 제약조건을 생성한다.

⑱ 객체 클래스의 전달 작용을 위한 객체 타입에 메소드를 생성한다.

(그림 12)에서 보여준 XML DTD로부터 변환된 객체모델 (그림 13)은 위에 설명된 내용에 따라 아래의 객체-관계형 데이터베이스 스키마로 변환되어 테이블이 생성된다.

(1) Partner 객체는 ProductCatalog 데이블에서 위에 열거된 ⑭번의 성질에 따라 (그림 15)와 같이 네스티드 테이블로 변환된다.

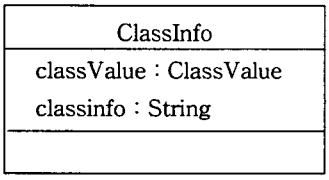



(그림 14) Partner 객체

```
SQL> CREATE TYPE Partner_t AS OBJECT
(
 name          String,
 partnerID     String,
 relationShip  String,
 role          String
);
SQL> CREATE TYPE Partner_list AS TABLE
                                OF Partner_t;
```

(그림 15) Partner 테이블

(2) ClassInfo 객체는 ProductCatalog 테이블에서 위에 열거된 ⑨번의 성질에 따라 부 클래스를 가지고 있으며, 또한 (그림 17)과 같이 네스티드 테이블로 변환된다.

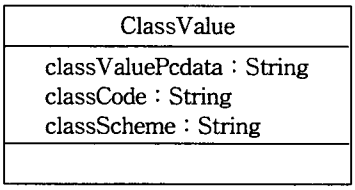



(그림 16) ClassInfo 객체

```
SQL> CREATE TYPE ClassInfo_t AS OBJECT
(
 classValue     ClassValue_t,
 classinfo      String
);
SQL> CREATE TYPE ClassInfo_list AS TABLE
                                OF ClassInfo_t;
```

(그림 17) ClassInfo 테이블

아래의 변환은 부 클래스로써 ClassValue 객체가 생성되고 위에 설명된 ClassInfo 테이블의 속성이 되어 (그림 19)와 같이 ClassValue 타입이 된다.

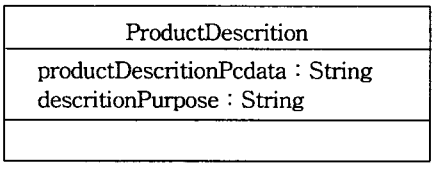



(그림 18) ClassValue 객체

```
SQL> CREATE TYPE ClassValue_t AS OBJECT
(
 classValuePcdata    String,
 classCode           String,
 classScheme         String
);
SQL> CREATE TABLE ClassValue OF
                              ClassValue_t;
```

(그림 19) ClassValue 타입

(3) ProductDescrition 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 (그림 21)과 같이 네스티드 테이블로 변환된다.




(그림 20) ProductDescription 객체

```
SQL> CREATE TYPE ProductDescrition_t AS
                                    OBJECT
(
 productDescritionPcdata     String,
 descritionPurpose           String
);
SQL> CREATE TYPE ProductDescrition_list AS
                TABLE OF ProductDescrition_t;
```

(그림 21) ProductDescription 테이블

(4) Manufacturer 객체는 ProductCatalog 테이블에서 위에 열거된 ⑤번의 성질에 따라 테이블 속성에 포함되어 (그림 23)과 같이 변환된다.

| Manufacture |
| --- |
| manufacturePcdata : String<br>partnerRef : String |
| |

(그림 22) Manufacture 객체

```
SQL> CREATE TYPE Manufacture_t AS
                                    OBJECT
(
 manufacturePcdata    String,
 partnerRef           String
);
SQL> CREATE TABLE Manufacture OF
                              Manufacture_t;
```

(그림 23) Manufacturer 테이블

(5) Participants 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 (그림 25)와 같이 네스티드 테이블로 변환된다.

| Participants |
| --- |
| participantsPcdata : String<br>partnerRef : String |
| |

(그림 24) Participants 객체

```
SQL> CREATE TYPE Participants_t AS OBJECT
(
 participantsPcdata     String,
 partnerRef             String
);
SQL> CREATE TYPE Participants_list AS
                    TABLE  OF Participants_t;
```

(그림 25) Participants 테이블

(6) ProductPrice 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 (그림 27)과 같이 네스티드 테이블로 변환된다.

| ProductPrice |
| --- |
| amount : String,<br>buyer : String |
| |

(그림 26) ProductPrice 객체

```
SQL> CREATE TYPE ProductPrice_t AS OBJECT
(
 amount              String,
 buyer               String
);
SQL> CREATE TYPE ProductPrice_list AS
                    TABLE OF ProductPrice_t;
```

(그림 27) ProductPrice 데이블

(7) Specification 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 (그림 29)와 같이 네스티드 테이블로 변환된다.

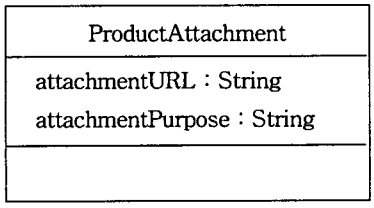
| Specification |
| --- |
| specificationPcdata : String<br>typeCode : String<br>uom : String<br>specificationName : String |
| |

(그림 28) Specification 객체

```
SQL> CREATE TYPE Specification_t AS OBJECT
(
 specificationPcdata        String,
 typeCode                   String,
 uom                        String,
 specificationName          String
);
SQL> CREATE TYPE Specification_list AS
                    TABLE OF Specification_t,
```

(그림 29) Specification 테이블

(8) ProductAttachment 객체는 ProductCatalog 테이블에서 위에 열거된 ⑤번의 성질에 따라 테이블 속성에 포함되어 (그림 31)과 같이 변환된다.




(그림 30) ProductAttachment 객체

```
SQL> CREATE TYPE ProductAttachment_t AS
                                    OBJECT
(
 attachmentURL          String,
 attachmentPurpose      String
);
SQL> CREATE TABLE ProductAttachment OF
                            ProductAttachment_t;
```

(그림 31) ProductAttachment 테이블

위에서 변환된 각각의 객체들은 기본 테이블 스키마로 아래와 같이 구성되어 (그림 32)와 같이 타입이 생성되고, (그림 33)과 같이 테이블이 생성된다.

```
SQL> CREATE TYPE ProductCatalog_t AS
        OBJECT
(
 productID              String,
 productName            String,
 defaultLanguage        String,
 defaultCurrency        String,
 countryOfOrigin        String,
 partner                Partner_list,
 classInfo              ClassInfo_list,
 productDescrition      ProductDescrition_list,
 manufacturer           Manufacturer_t,
 participants           Participants_list,
 productPrice           ProductPrice_list,
 specification          Specification_list,
 productAttachment      ProductAttachment_t
);
```

(그림 32) ProductCatalog 타입

```
SQL> CREATE TABLE ProductCatalog OF
            ProductCatalog_t
        NESTED TABLE partner STORE AS
            Partner_tbl;
        NESTED TABLE classInfo STORE AS
            ClassInfo_tbl;
        NESTED TABLE productDescrition
            STORE AS ProductDescrition_tbl;
        NESTED TABLE participants STORE AS
            Participants_tbl;
        NESTED TABLE productPrice STORE
            AS ProductPrice_tbl;
        NESTED TABLE specification STORE AS
            Specification_tbl;
```

(그림 33) ProductCatalog 테이블

## 6. 결론 및 향후 연구방향

XML은 웹에서 구조화된 정보나 반-구조화된 정보를 교환하기 위한 표준 마크업 언어로 채택되어 가고 있다. 한편 웹에서 정보교환을

위한 XML 메시지의 소스 데이터는 Legacy 데이터베이스에 저장되어 있기 때문에, 이에 따라 XML 응용과 데이터베이스 시스템과의 원활한 연계가 요구되어진다. Oracle8i와 9i 및 Informix 그리고 SQL2000 등과 같이, DBMS 공급자들은 XML을 취급하기 위하여 자신의 DBMS들을 확장하고 있다. 이러한 방식의 XML 응용과 데이터베이스 시스템과의 연계방법은 플랫폼-종속적이며 아울러 DBMS-종속적이다. 따라서 보다 원활한 XML 응용과 데이터베이스 시스템과의 연계를 위해서는 플랫폼-독립적이며 DBMS-독립적인 연계방안이 요망된다.

XML 응용과 데이터베이스 시스템 사이의 원활한 연계를 위해서, XML DTD를 관계형 데이터베이스 스키마로 변환하는 방법에 대해서는 많은 연구가 진행되었으나, XML DTD를 객체-관계형 데이터베이스 스키마로 변환키 위한 연구는 미미한 실정이다. 멀티미디어 응용 등을 위하여 기존의 관계형 DBMS들이 객체-관계형 DBMS로 확장된 현실에서 XML DTD를 객체-관계형 데이터베이스 스키마로 변환키 위한 연구는 조속히 요구되고 있다.

본 논문에서는 XML DTD를 객체-관계형 데이터베이스 스키마로 자동 변환되게 함으로써 XML 응용과 객체-관계형 데이터베이스 시스템 사이에 원활한 연계가 이루어지도록 하였다.

향후 연구 방향으로 XML DTD를 객체-관계형 데이터베이스 스키마로 자동변환키 위한 미들웨어 컴포넌트들을 CDB(Component Based Design) 방법에 따라 추출하고 설계하여 이를 EJB로 구현하고자 한다.

## 참 고 문 헌

[1] Andrew Davidson, Matthew Fuchs, Mette Hedin, Mudia Jain, Jari Koistinen, Chris Lloyd, Murray Maloney, Kelly Schwarzhof, "Schema for Object-Oriented XML 2.0," July, 1999. See http://www.w3.org/TR/NOTE-SOX.

[2] Hiroshi Matuyama; Kent Tamura; Naohiko Uramoto, XML and Java Developing Web Applications, Addison Wesley, 1999, pp.20-21.

[3] Joo Kyung-soo, "A Design of Middleware Components for the Connection between XML and RDB," 2001 IEEE International Symposium on Industrial Electronics Proceedings, Pusan, Korea, June, 2001.

[4] Michel Goosens, Janne Saarela, "A Practical Introduction to SGML," volume 16(3), In TUGboat, 1995.

[5] World Wide Wed Consortium, "The XML Data Model," http://www.w3.org/XML/Datamodel.html, 2000.

[6] 이상태, 주경수, "제약조건 유지를 위한 XML DTD의 관계 스키마로 변환 방법", 한국인터넷정보학회, 제1권, 제2호, pp.189-196, 2000.

[7] 이상태, 주경수. "UML 객체모델의 ORDB 객체 매핑", 한국인터넷정보학회, 제2권, 제1호, pp.187-191, 2001.

[8] 이상태, 주경수. "객체모델을 이용한 XML DTD의 ORDB 스키마로의 변환", 한국데이터베이스학회, 춘계 Conference, pp.303-310, 2001.

[9] 이상태, 이정수, 주경수. "객체모델을 기반으로 한 XML DTD의 RDB 스키마로의 변환 방법", 대한전자공학회, 하계종합학술대회 논문집III, pp.113-116, 2001.

■ 저자소개

이 상 태

건국대학교를 졸업하고, 독일 DORTMUND대학교에서 전산학 석사학위를 취득하였으며, 순천향대학교 대학원 전산학과 박사과정을 수료하였고, 현재 신성대학 컴퓨터응용계열 교수로 재직하고 있으며, 주요관심분야는 XML, 분산데이터베이스, 전자상거래, 인터넷 응용분야 등이다.

주 경 수

고려대학교를 졸업하였으며, 동 대학원에서 이학석사, 이학박사 학위를 취득하였고, 현재 순천향대학교 정보기술공학부 정교수로 재직하고 있으며, 주요관심분야는 Database Systems, System Integration, Object-oriented Systems 등이다.